Objective:

Understand the Dataset & cleanup (if required). Build Regression models to predict the sales w.r.t a single & multiple feature. Also evaluate the models & compare their respective scores like R2, RMSE, etc.

Stractegic Plan of Action:

We aim to solve the problem statement by creating a plan of action, Here are some of the necessary steps:

1.Data Exploration

2.Exploratory Data Analysis (EDA)

3.Data Pre-processing

4.Data Manipulation

5.Feature Selection/Extraction

6.Predictive Modelling

7.Project Outcomes & Conclusion

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
#%matplotlib inline
plt.rcParams['figure.figsize'] = [10,6]
import warnings
warnings.filterwarnings('ignore')

from IPython.display import display

#from brokenaxes import brokenaxes
from statsmodels.formula import api
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import
variance_inflation_factor

from sklearn.decomposition import PCA
from sklearn.linear_model import
Ridge,Lasso,ElasticNet,LinearRegression
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error
```

1. Data Exploration

```
walmartdf = pd.read_csv('walmart.csv')
```

```
walmartdf.head()
```

```
   Store        Date  Weekly_Sales  Holiday_Flag  Temperature
Fuel_Price  \
0      1  05-02-2010    1643690.90             0        42.31
2.572
1      1  12-02-2010    1641957.44             1        38.51
2.548
2      1  19-02-2010    1611968.17             0        39.93
2.514
3      1  26-02-2010    1409727.59             0        46.63
2.561
4      1  05-03-2010    1554806.68             0        46.50
2.625

          CPI  Unemployment
0  211.096358         8.106
1  211.242170         8.106
2  211.289143         8.106
3  211.319643         8.106
4  211.350143         8.106
```

```
walmartdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   object
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: float64(5), int64(2), object(1)
memory usage: 402.3+ KB
```

```
walmartdf.describe()
```

```
             Store  Weekly_Sales  Holiday_Flag  Temperature
Fuel_Price  \
count  6435.000000  6.435000e+03   6435.000000  6435.000000
```

```
6435.000000
mean      23.000000  1.046965e+06      0.069930    60.663782
3.358607
std       12.988182  5.643666e+05      0.255049    18.444933
0.459020
min        1.000000  2.099862e+05      0.000000    -2.060000
2.472000
25%       12.000000  5.533501e+05      0.000000    47.460000
2.933000
50%       23.000000  9.607460e+05      0.000000    62.670000
3.445000
75%       34.000000  1.420159e+06      0.000000    74.940000
3.735000
max       45.000000  3.818686e+06      1.000000   100.140000
4.468000

               CPI  Unemployment
count  6435.000000   6435.000000
mean    171.578394      7.999151
std      39.356712      1.875885
min     126.064000      3.879000
25%     131.735000      6.891000
50%     182.616521      7.874000
75%     212.743293      8.622000
max     227.232807     14.313000
```

```
walmartdf.shape
```

```
(6435, 8)
```

Inference: The Datset consists of 8 features & 6435 samples.

```
walmartdf.Date=pd.to_datetime(walmartdf.Date)

walmartdf['weekday'] = walmartdf.Date.dt.weekday
walmartdf['month'] = walmartdf.Date.dt.month
walmartdf['year'] = walmartdf.Date.dt.year

# df['Monthly_Quarter'] =
df.month.map({1:'Q1',2:'Q1',3:'Q1',4:'Q2',5:'Q2',6:'Q2',7:'Q3',
#
8:'Q3',9:'Q3',10:'Q4',11:'Q4',12:'Q4'})

walmartdf.drop(['Date'], axis=1, inplace=True)#,'month'

target = 'Weekly_Sales'
features = [i for i in walmartdf.columns if i not in [target]]
original_walmartdf = walmartdf.copy(deep=True)

walmartdf.head()
```

```
   Store  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  
CPI  \
0      1    1643690.90             0        42.31       2.572  
211.096358   
1      1    1641957.44             1        38.51       2.548  
211.242170   
2      1    1611968.17             0        39.93       2.514  
211.289143   
3      1    1409727.59             0        46.63       2.561  
211.319643   
4      1    1554806.68             0        46.50       2.625  
211.350143   

   Unemployment  weekday  month  year  
0         8.106        6      5  2010  
1         8.106        3     12  2010  
2         8.106        4      2  2010  
3         8.106        4      2  2010  
4         8.106        0      5  2010  
```

```
walmartdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         6435 non-null   int64  
 1   Weekly_Sales  6435 non-null   float64
 2   Holiday_Flag  6435 non-null   int64  
 3   Temperature   6435 non-null   float64
 4   Fuel_Price    6435 non-null   float64
 5   CPI           6435 non-null   float64
 6   Unemployment  6435 non-null   float64
 7   weekday       6435 non-null   int64  
 8   month         6435 non-null   int64  
 9   year          6435 non-null   int64  
dtypes: float64(5), int64(5)
memory usage: 502.9 KB
```

```
#Checking number of unique rows in each feature

walmartdf.nunique().sort_values()
```

```
Holiday_Flag       2
year               3
weekday            7
month             12
Store             45
Unemployment     349
Fuel_Price       892
```

```
CPI              2145
Temperature      3528
Weekly_Sales     6435
dtype: int64
```

```
#Checking number of unique rows in each feature

nu = walmartdf[features].nunique().sort_values()
nf = []; cf = []; nnf = 0; ncf = 0; #numerical & categorical features

for i in range(walmartdf[features].shape[1]):
    if nu.values[i]<=45:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\n\033[1mInference:\033[0m The Datset has {} numerical & {}
categorical features.'.format(len(nf),len(cf)))
```

```
Inference: The Datset has 4 numerical & 5 categorical features.
```

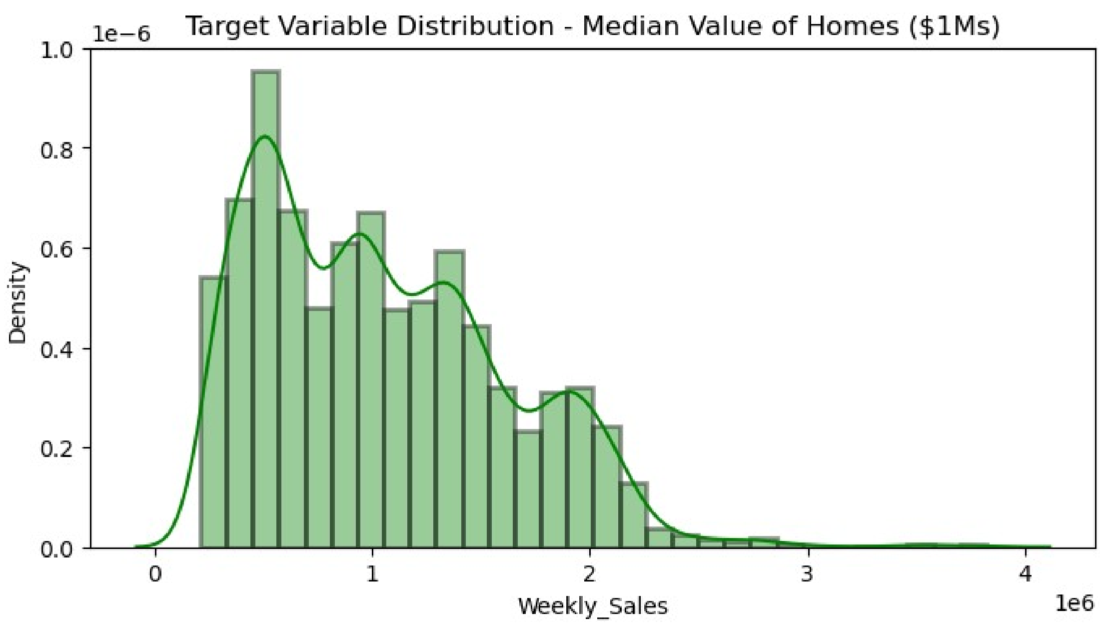
```
walmartdf.describe()
```

```
             Store  Weekly_Sales  Holiday_Flag  Temperature
Fuel_Price  \
count  6435.000000  6.435000e+03   6435.000000  6435.000000
6435.000000
mean     23.000000  1.046965e+06      0.069930    60.663782
3.358607
std      12.988182  5.643666e+05      0.255049    18.444933
0.459020
min       1.000000  2.099862e+05      0.000000    -2.060000
2.472000
25%      12.000000  5.533501e+05      0.000000    47.460000
2.933000
50%      23.000000  9.607460e+05      0.000000    62.670000
3.445000
75%      34.000000  1.420159e+06      0.000000    74.940000
3.735000
max      45.000000  3.818686e+06      1.000000   100.140000
4.468000

               CPI  Unemployment      weekday        month
year
count  6435.000000   6435.000000  6435.000000  6435.000000
6435.000000
mean    171.578394      7.999151     3.573427     6.475524
2010.965035
std      39.356712      1.875885     1.426581     3.321797
0.797019
min     126.064000      3.879000     0.000000     1.000000
2010.000000
```

```
25%     131.735000      6.891000     4.000000     4.000000
2010.000000
50%     182.616521      7.874000     4.000000     6.000000
2011.000000
75%     212.743293      8.622000     4.000000     9.000000
2012.000000
max     227.232807     14.313000     6.000000    12.000000
2012.000000
```

2. Exploratory Data Analysis (EDA)

```
#Let us first analyze the distribution of the target variable

plt.figure(figsize=[8,4])
sns.distplot(walmartdf[target],
color='g',hist_kws=dict(edgecolor="black", linewidth=2), bins=30)
plt.title('Target Variable Distribution - Median Value of Homes
($1Ms)')
plt.show()
```

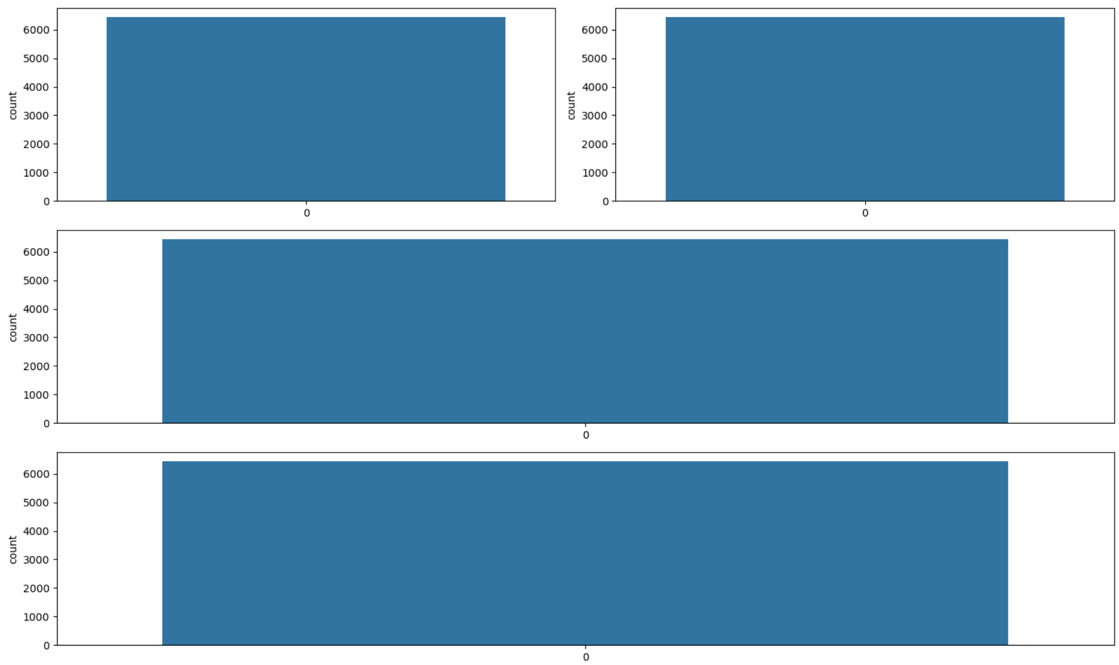


The Target Variable seems to be be normally distributed, averaging around 20 units.

```
#Visualising the categorical features

print('\033[1mVisualising Categorical Features:'.center(100))

n=2
clr=['r','g','b','g','b','r']
plt.figure(figsize=[15,3*math.ceil(len(cf)/n)])

for i in range(len(cf)):
    if walmartdf[cf[i]].nunique()<=8:
        plt.subplot(math.ceil(len(cf)/n),n,i+1)
```

```
        sns.countplot(walmartdf[cf[i]])
    else:
        plt.subplot(3,1,i-1)
        sns.countplot(walmartdf[cf[i]])

plt.tight_layout()
plt.show()
```

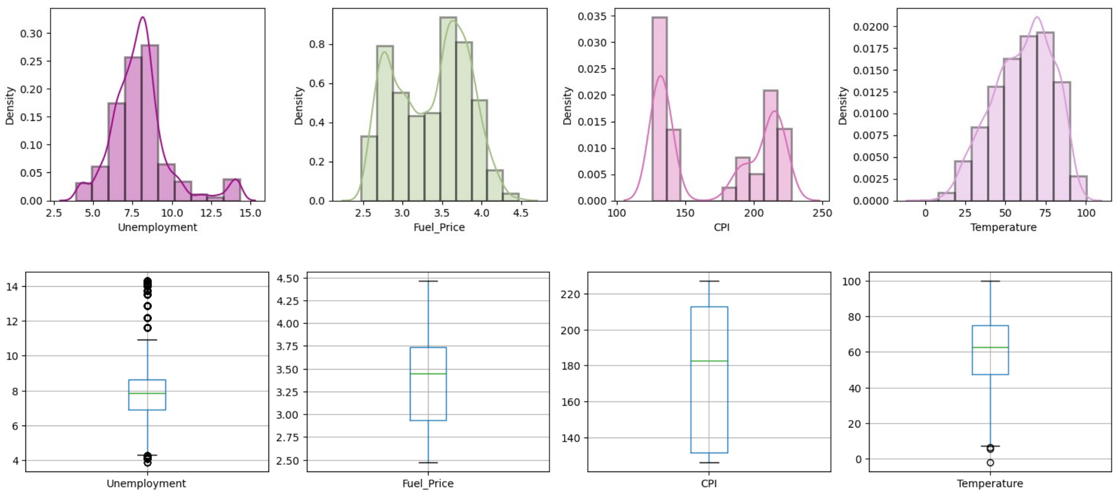
Visualising Categorical Features:




```
#Visualising the numeric features

print('\033[1mNumeric Features Distribution'.center(130))

n=4

clr=['r','g','b','g','b','r']

plt.figure(figsize=[15,6*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    sns.distplot(walmartdf[nf[i]],hist_kws=dict(edgecolor="black",
linewidth=2), bins=10,
color=list(np.random.randint([255,255,255])/255))
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,6*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
```

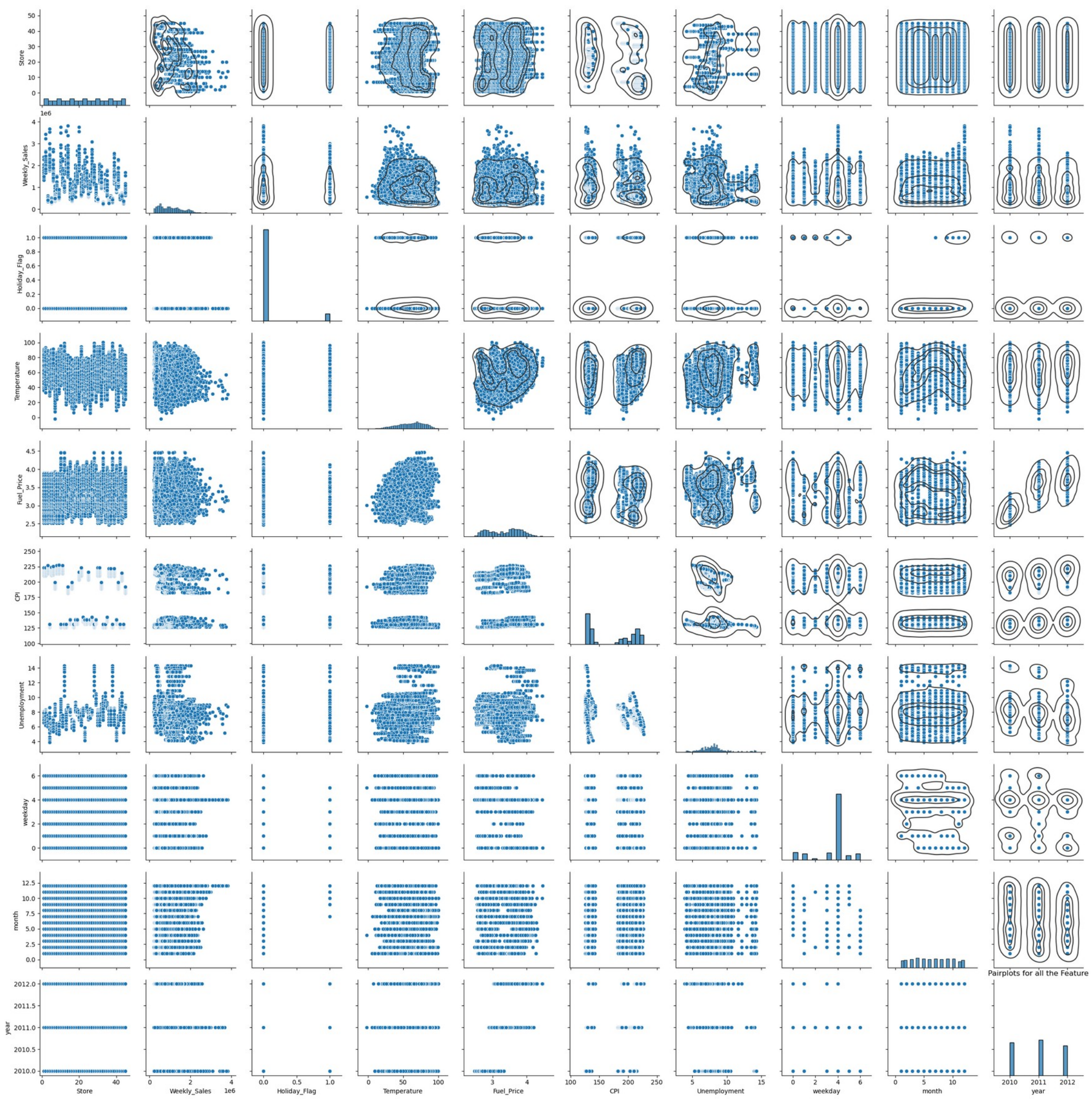
```
    walmartdf.boxplot(nf[i])
plt.tight_layout()
plt.show()
```

Numeric Features
Distribution



There seem to be some outliers

```
#Understanding the relationship between all the features

g = sns.pairplot(walmartdf)
plt.title('Pairplots for all the Feature')
g.map_upper(sns.kdeplot, levels=4, color=".2")
plt.show()
```



We can notice that some features have linear relationship, let us futher analyze the detect multicollinearity.

3. Data Preprocessing

```
#Removal of any Duplicate rows (if any)

counter = 0
rs,cs = original_walmartdf.shape

walmartdf.drop_duplicates(inplace=True)

if walmartdf.shape==(rs,cs):
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any 
duplicates')
else:
```

```
    print(f'\n\033[1mInference:\033[0m Number of duplicates
dropped/fixed ---> {rs-df.shape[0]}')
```

```
Inference: The dataset doesn't have any duplicates
```

```
#Check for empty elements

nvc = pd.DataFrame(walmartdf.isnull().sum().sort_values(),
columns=['Total Null Values'])
nvc['Percentage'] = round(nvc['Total Null
Values']/walmartdf.shape[0],3)*100
print(nvc)
```

```
              Total Null Values  Percentage
Store                         0         0.0
Weekly_Sales                  0         0.0
Holiday_Flag                  0         0.0
Temperature                   0         0.0
Fuel_Price                    0         0.0
CPI                           0         0.0
Unemployment                  0         0.0
weekday                       0         0.0
month                         0         0.0
year                          0         0.0
```

```
#Converting categorical Columns to Numeric

df = walmartdf.copy()

ecc = nvc[nvc['Percentage']!=0].index.values
fcc = [i for i in cf if i not in ecc]
#One-Hot Binay Encoding
oh=True
dm=True
for i in fcc:
    #print(i)
    if df[i].nunique()==2:
        if oh==True: print("\033[1mOne-Hot Encoding on features:\
033[0m")
        print(i);oh=False
        df[i]=pd.get_dummies(df[i], drop_first=True, prefix=str(i))
    if (df[i].nunique()>2):
        if dm==True: print("\n\033[1mDummy Encoding on features:\
033[0m")
        print(i);dm=False
        df = pd.concat([df.drop([i], axis=1),
pd.DataFrame(pd.get_dummies(df[i], drop_first=True,
prefix=str(i)))],axis=1)
```

```
df.shape
```

```
One-Hot Encoding on features:
Holiday_Flag

Dummy Encoding on features:
year
weekday
month
Store

(6435, 69)
```

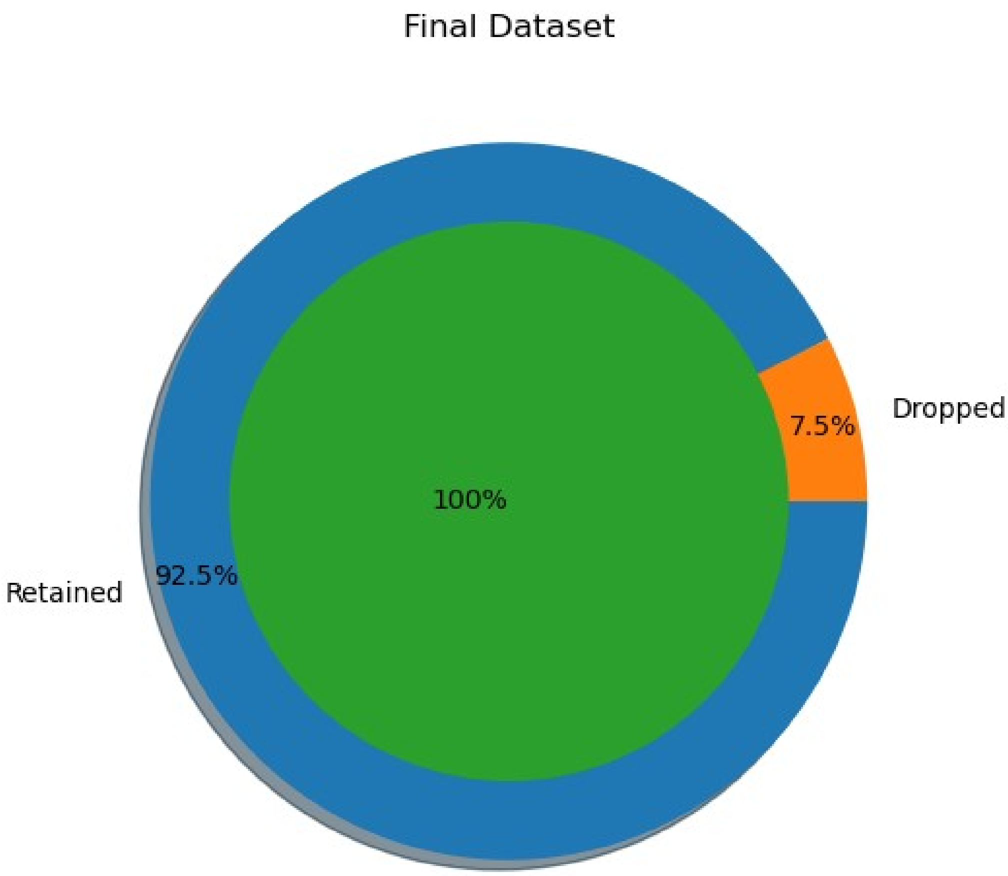
```
#Removal of outlier:

df1 = df.copy()

#features1 = [i for i in features if i not in ['CHAS','RAD']]
features1 = nf

for i in features1:
    Q1 = df1[i].quantile(0.25)
    Q3 = df1[i].quantile(0.75)
    IQR = Q3 - Q1
    df1 = df1[df1[i] <= (Q3+(1.5*IQR))]
    df1 = df1[df1[i] >= (Q1-(1.5*IQR))]
    df1 = df1.reset_index(drop=True)
display(df1.head())
print('\n\033[1mInference:\033[0m\nBefore removal of outliers, The
dataset had {} samples.'.format(df.shape[0]))
print('After removal of outliers, The dataset now has {}
samples.'.format(df1.shape[0]))
```

```
   Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price         CPI  \
0    1643690.90             0        42.31       2.572  211.096358
1    1641957.44             1        38.51       2.548  211.242170
2    1611968.17             0        39.93       2.514  211.289143
3    1409727.59             0        46.63       2.561  211.319643
4    1554806.68             0        46.50       2.625  211.350143

   Unemployment  year_2011  year_2012  weekday_1  weekday_2  ...
Store_36  \
0         8.106          0          0          0          0  ...
0
1         8.106          0          0          0          0  ...
0
2         8.106          0          0          0          0  ...
0
3         8.106          0          0          0          0  ...
0
```

```
4         8.106          0          0          0          0  ...
0

   Store_37  Store_38  Store_39  Store_40  Store_41  Store_42
Store_43  \
0         0         0         0         0         0         0
0
1         0         0         0         0         0         0
0
2         0         0         0         0         0         0
0
3         0         0         0         0         0         0
0
4         0         0         0         0         0         0
0

   Store_44  Store_45
0         0         0
1         0         0
2         0         0
3         0         0
4         0         0

[5 rows x 69 columns]


Inference:
Before removal of outliers, The dataset had 6435 samples.
After removal of outliers, The dataset now has 5953 samples.
```

```
#Final Dataset size after performing Preprocessing

final_df = df1.copy()
final_df.columns=[i.replace('-','_') for i in df.columns]

plt.title('Final Dataset')
plt.pie([final_df.shape[0], original_walmartdf.shape[0]-
final_df.shape[0]], radius = 1, labels=['Retained','Dropped'],
counterclock=False,
        autopct='%1.1f%%', pctdistance=0.9, explode=[0,0],
shadow=True)
plt.pie([final_df.shape[0]], labels=['100%'], labeldistance=-0,
radius=0.78)
plt.show()

print(f'\n\033[1mInference:\033[0m After the cleanup process,
{original_walmartdf.shape[0]-final_df.shape[0]} samples were dropped,
\
while retaining {round(100 -
```

```
(final_df.shape[0]*100/(original_walmartdf.shape[0])),2)}% of the
data.')
```



Inference: After the cleanup process, 482 samples were dropped, while retaining 7.49% of the data.

4. Data Manipulation

```
#Splitting the data intro training & testing sets

m=[]
for i in df.columns.values:
    m.append(i.replace(' ','_'))

df.columns = m
X = final_df.drop([target],axis=1)
Y = final_df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y,
train_size=0.8, test_size=0.2, random_state=100)
Train_X.reset_index(drop=True,inplace=True)
```

```
print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  --->
',Train_X.shape,Train_Y.shape,'\nTesting set   ---> ',
Test_X.shape,'', Test_Y.shape)
```

```
Original set  --->  (5953, 68) (5953,)
Training set  --->  (4762, 68) (4762,)
Testing set   --->  (1191, 68)  (1191,)
```

```
#Feature Scaling (Standardization)

std = StandardScaler()

print('\033[1mStandardardization on Training set'.center(120))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n','\033[1mStandardardization on Testing set'.center(120))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())
```

```
                                         Standardardization on
Training set

       Holiday_Flag   Temperature    Fuel_Price           CPI
Unemployment  \
count  4.762000e+03  4.762000e+03  4.762000e+03  4.762000e+03
4.762000e+03
mean  -1.492110e-18 -1.305596e-16 -2.991680e-16 -2.762268e-16 -
4.267434e-16
std    1.000105e+00  1.000105e+00  1.000105e+00  1.000105e+00
1.000105e+00
min   -2.742012e-01 -2.961575e+00 -1.871814e+00 -1.248731e+00 -
2.762670e+00
25%   -2.742012e-01 -7.314248e-01 -9.886990e-01 -1.076949e+00 -
6.783836e-01
50%   -2.742012e-01  1.062547e-01  1.663112e-01  3.842133e-01
9.596435e-02
75%   -2.742012e-01  7.731979e-01  8.427860e-01  9.933828e-01
6.138095e-01
max    3.646958e+00  2.170008e+00  2.469806e+00  1.340791e+00
2.575491e+00

          year_2011     year_2012     weekday_1     weekday_2
weekday_3  \
count  4.762000e+03  4.762000e+03  4.762000e+03  4.762000e+03
4.762000e+03
mean  -5.520807e-17  1.939743e-17  4.513632e-17  1.492110e-18 -
3.245339e-17
std    1.000105e+00  1.000105e+00  1.000105e+00  1.000105e+00
```

```
1.000105e+00
min   -7.526270e-01 -6.371530e-01 -2.588345e-01 -1.157891e-01 -
2.719813e-01
25%   -7.526270e-01 -6.371530e-01 -2.588345e-01 -1.157891e-01 -
2.719813e-01
50%   -7.526270e-01 -6.371530e-01 -2.588345e-01 -1.157891e-01 -
2.719813e-01
75%    1.328679e+00  1.569482e+00 -2.588345e-01 -1.157891e-01 -
2.719813e-01
max    1.328679e+00  1.569482e+00  3.863473e+00  8.636394e+00
3.676723e+00

       ...      Store_36      Store_37      Store_38      Store_39  \
count  ...  4.762000e+03  4.762000e+03  4.762000e+03  4.762000e+03
mean   ...  1.529413e-17  4.923963e-17  8.952659e-18  1.342899e-17
std    ...  1.000105e+00  1.000105e+00  1.000105e+00  1.000105e+00
min    ... -1.573123e-01 -1.587085e-01 -4.351484e-02 -1.580118e-01
25%    ... -1.573123e-01 -1.587085e-01 -4.351484e-02 -1.580118e-01
50%    ... -1.573123e-01 -1.587085e-01 -4.351484e-02 -1.580118e-01
75%    ... -1.573123e-01 -1.587085e-01 -4.351484e-02 -1.580118e-01
max    ...  6.356783e+00  6.300861e+00  2.298067e+01  6.328643e+00

           Store_40      Store_41      Store_42      Store_43
Store_44  \
count  4.762000e+03  4.762000e+03  4.762000e+03  4.762000e+03
4.762000e+03
mean  -3.730275e-18 -1.492110e-17  3.730275e-17 -2.984220e-17 -
4.774752e-17
std    1.000105e+00  1.000105e+00  1.000105e+00  1.000105e+00
1.000105e+00
min   -1.307162e-01 -1.537717e-01 -1.573123e-01 -1.628322e-01 -
1.551967e-01
25%   -1.307162e-01 -1.537717e-01 -1.573123e-01 -1.628322e-01 -
1.551967e-01
50%   -1.307162e-01 -1.537717e-01 -1.573123e-01 -1.628322e-01 -
1.551967e-01
75%   -1.307162e-01 -1.537717e-01 -1.573123e-01 -1.628322e-01 -
1.551967e-01
max    7.650163e+00  6.503146e+00  6.356783e+00  6.141290e+00
6.443435e+00

           Store_45
count  4.762000e+03
mean   2.499284e-17
std    1.000105e+00
min   -1.523346e-01
25%   -1.523346e-01
50%   -1.523346e-01
75%   -1.523346e-01
max    6.564495e+00
```

[8 rows x 68 columns]

Standardardization on Testing set

	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment \
count	1191.000000	1191.000000	1191.000000	1191.000000	1191.000000
mean	0.005646	0.044406	0.075113	0.021041	-0.050953
std	1.009885	1.000220	0.971917	1.004644	1.010206
min	-0.274201	-2.857425	-1.780457	-1.248731	-2.762670
25%	-0.274201	-0.657516	-0.852751	-1.077025	-0.699355
50%	-0.274201	0.187351	0.298996	0.393492	0.058860
75%	-0.274201	0.818764	0.844961	1.019967	0.611390
max	3.646958	2.035481	2.469806	1.345814	2.575491

	year_2011	year_2012	weekday_1	weekday_2	weekday_3	... \
count	1191.000000	1191.000000	1191.000000	1191.000000	1191.000000	...
mean	0.052984	0.065042	0.007679	0.038532	0.026409	...
std	1.014188	1.028250	1.014142	1.152364	1.044095	...
min	-0.752627	-0.637153	-0.258834	-0.115789	-0.271981	...
25%	-0.752627	-0.637153	-0.258834	-0.115789	-0.271981	...
50%	-0.752627	-0.637153	-0.258834	-0.115789	-0.271981	...
75%	1.328679	1.569482	-0.258834	-0.115789	-0.271981	...
max	1.328679	1.569482	3.863473	8.636394	3.676723	...

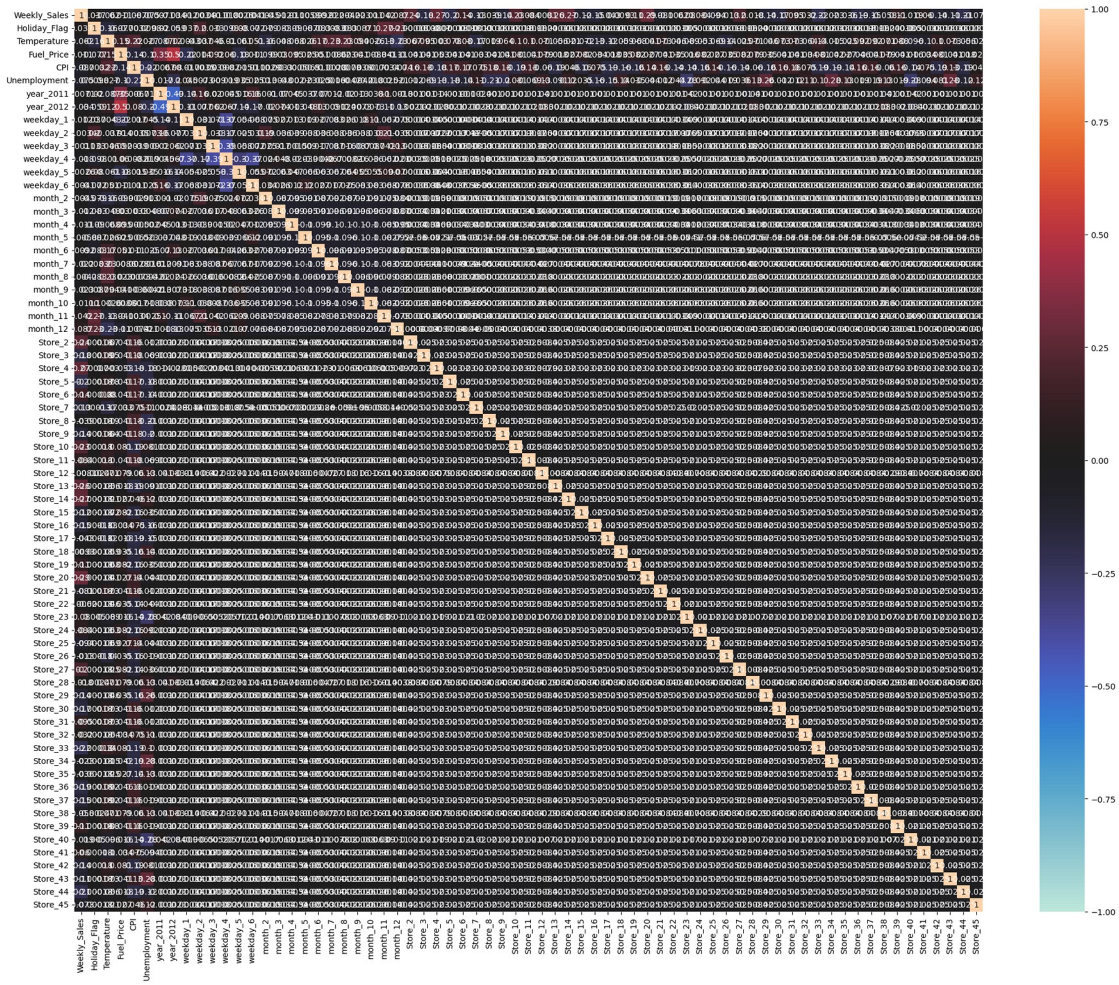
	Store_36	Store_37	Store_38	Store_39	Store_40 \
count	1191.000000	1191.000000	1191.000000	1191.000000	1191.000000
mean	-0.004168	-0.017693	0.111140	-0.010959	-0.000055

```
std        0.987401     0.944330     1.881448     0.965939     1.000214
min       -0.157312    -0.158708    -0.043515    -0.158012    -0.130716
25%       -0.157312    -0.158708    -0.043515    -0.158012    -0.130716
50%       -0.157312    -0.158708    -0.043515    -0.158012    -0.130716
75%       -0.157312    -0.158708    -0.043515    -0.158012    -0.130716
max        6.356783     6.300861    22.980668     6.328643     7.650163

              Store_41     Store_42     Store_43     Store_44     Store_45
count   1191.000000  1191.000000  1191.000000  1191.000000  1191.000000
mean       0.030677    -0.004168    -0.056970     0.016556     0.045053
std        1.093088     0.987401     0.810380     1.051077     1.134875
min       -0.153772    -0.157312    -0.162832    -0.155197    -0.152335
25%       -0.153772    -0.157312    -0.162832    -0.155197    -0.152335
50%       -0.153772    -0.157312    -0.162832    -0.155197    -0.152335
75%       -0.153772    -0.157312    -0.162832    -0.155197    -0.152335
max        6.503146     6.356783     6.141290     6.443435     6.564495

[8 rows x 68 columns]
```

5. Feature Selection/Extraction

```
#Checking the correlation

print('\033[1mCorrelation Matrix'.center(100))
plt.figure(figsize=[25,20])
sns.heatmap(final_df.corr(), annot=True, vmin=-1, vmax=1, center=0)
#cmap='BuGn'
plt.show()
```

```
                                       Correlation Matrix
```



There seems to be strong multi-correlation between the features.

```
#Testing a Linear Regression model with statsmodels

Train_xy =
pd.concat([Train_X_std,Train_Y.reset_index(drop=True)],axis=1)
a = Train_xy.columns.values

API = api.ols(formula='{} ~ {}'.format(target,' + '.join(i for i in
Train_X.columns)), data=Train_xy).fit()
#print(API.conf_int())
#print(API.pvalues)
API.summary()

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

======================================================================
========
Dep. Variable:           Weekly_Sales   R-squared:
```

```
0.933
Model:                            OLS   Adj. R-squared:
0.932
Method:                 Least Squares   F-statistic:
958.4
Date:                Mon, 03 Apr 2023   Prob (F-statistic):
0.00
Time:                        22:18:29   Log-Likelihood:
-63430.
No. Observations:                4762   AIC:
1.270e+05
Df Residuals:                    4693   BIC:
1.274e+05
Df Model:                          68

Covariance Type:            nonrobust

======================================================================
==========
                   coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
----------
Intercept     1.048e+06   2152.234    486.752      0.000    1.04e+06
1.05e+06
Holiday_Flag  3435.7934   2436.850      1.410      0.159   -1341.577
8213.164
Temperature  -1.091e+04   3611.329     -3.022      0.003    -1.8e+04
-3834.746
Fuel_Price    4743.1404   6083.604      0.780      0.436   -7183.581
1.67e+04
CPI           4.806e+05   6.53e+04      7.359      0.000    3.53e+05
6.09e+05
Unemployment -6.096e+04   7305.771     -8.344      0.000   -7.53e+04
-4.66e+04
year_2011    -3.085e+04   6385.220     -4.832      0.000   -4.34e+04
-1.83e+04
year_2012    -7.005e+04   8794.662     -7.965      0.000   -8.73e+04
-5.28e+04
weekday_1     5298.1880   3028.935      1.749      0.080    -639.948
1.12e+04
weekday_2    -1.144e+04   2552.678     -4.481      0.000   -1.64e+04
-6435.135
weekday_3    -1.397e+04   3035.015     -4.603      0.000   -1.99e+04
-8021.037
weekday_4    -1.668e+04   4056.274     -4.113      0.000   -2.46e+04
-8730.185
weekday_5    -1.288e+04   2827.562     -4.555      0.000   -1.84e+04
-7335.388
weekday_6    -2292.8321   3074.313     -0.746      0.456   -8319.930
```

```
3734.265
month_2        2.869e+04   3169.875      9.050      0.000    2.25e+04
3.49e+04
month_3        2.018e+04   3233.395      6.243      0.000    1.38e+04
2.65e+04
month_4        2.099e+04   3462.153      6.062      0.000    1.42e+04
2.78e+04
month_5        2.214e+04   3478.401      6.364      0.000    1.53e+04
2.9e+04
month_6        3.159e+04   3270.641      9.659      0.000    2.52e+04
3.8e+04
month_7        1.905e+04   3478.984      5.474      0.000    1.22e+04
2.59e+04
month_8        2.445e+04   3357.585      7.282      0.000    1.79e+04
3.1e+04
month_9        1.231e+04   3456.199      3.561      0.000    5531.812
1.91e+04
month_10       1.696e+04   3519.751      4.820      0.000    1.01e+04
2.39e+04
month_11       4.132e+04   3243.964     12.736      0.000     3.5e+04
4.77e+04
month_12       6.241e+04   3556.769     17.546      0.000    5.54e+04
6.94e+04
Store_2        5.522e+04   2977.883     18.545      0.000    4.94e+04
6.11e+04
Store_3       -1.841e+05   3122.600    -58.973      0.000    -1.9e+05
-1.78e+05
Store_4        2.173e+05   2.14e+04     10.161      0.000    1.75e+05
2.59e+05
Store_5       -1.914e+05   3140.260    -60.961      0.000   -1.98e+05
-1.85e+05
Store_6       -9300.0797   3167.398     -2.936      0.003   -1.55e+04
-3090.492
Store_7       -1.058e+05   6373.008    -16.605      0.000   -1.18e+05
-9.33e+04
Store_8        -1.21e+05   3457.789    -34.980      0.000   -1.28e+05
-1.14e+05
Store_9       -1.635e+05   3269.593    -50.014      0.000    -1.7e+05
-1.57e+05
Store_10        2.18e+05   2.21e+04      9.844      0.000    1.75e+05
2.61e+05
Store_11      -4.043e+04   3138.072    -12.885      0.000   -4.66e+04
-3.43e+04
Store_12       3.965e+04   8711.520      4.551      0.000    2.26e+04
5.67e+04
Store_13       2.231e+05   2.23e+04     10.021      0.000    1.79e+05
2.67e+05
Store_14       1.319e+05   8018.338     16.446      0.000    1.16e+05
1.48e+05
Store_15       8382.1132    2.1e+04      0.399      0.690   -3.28e+04
```

```
4.96e+04
Store_16      -1.298e+05   6615.254    -19.618      0.000   -1.43e+05
-1.17e+05
Store_17       5.146e+04   2.25e+04      2.285      0.022    7312.718
9.56e+04
Store_18       8.797e+04    2.1e+04      4.182      0.000    4.67e+04
1.29e+05
Store_19       1.336e+05   2.08e+04      6.439      0.000    9.29e+04
1.74e+05
Store_20       9.329e+04   3507.943     26.594      0.000    8.64e+04
1e+05
Store_21      -1.213e+05   2997.397    -40.462      0.000   -1.27e+05
-1.15e+05
Store_22       6.746e+04   2.02e+04      3.344      0.001    2.79e+04
1.07e+05
Store_23       8.742e+04   1.83e+04      4.780      0.000    5.16e+04
1.23e+05
Store_24        1.24e+05   2.05e+04      6.040      0.000    8.38e+04
1.64e+05
Store_25      -1.259e+05   3601.097    -34.969      0.000   -1.33e+05
-1.19e+05
Store_26       6.731e+04    2.1e+04      3.201      0.001    2.61e+04
1.09e+05
Store_27       1.831e+05   2.04e+04      8.995      0.000    1.43e+05
2.23e+05
Store_28       5.097e+04   8150.402      6.253      0.000     3.5e+04
6.69e+04
Store_29       1.131e+04   2.15e+04      0.527      0.598   -3.08e+04
5.34e+04
Store_30      -1.784e+05   3064.227    -58.205      0.000   -1.84e+05
-1.72e+05
Store_31      -2.275e+04   2978.182     -7.638      0.000   -2.86e+04
-1.69e+04
Store_32      -1.326e+04   6496.832     -2.041      0.041    -2.6e+04
-521.400
Store_33      -2.769e+04   2.25e+04     -1.228      0.220   -7.19e+04
1.65e+04
Store_34       9.036e+04   2.24e+04      4.031      0.000    4.64e+04
1.34e+05
Store_35       5.377e+04   1.96e+04      2.738      0.006    1.53e+04
9.23e+04
Store_36      -1.773e+05   3036.670    -58.397      0.000   -1.83e+05
-1.71e+05
Store_37      -1.568e+05   3053.641    -51.364      0.000   -1.63e+05
-1.51e+05
Store_38       7900.7310   6870.763      1.150      0.250   -5569.191
2.14e+04
Store_39      -1.129e+04   3038.994     -3.716      0.000   -1.73e+04
-5335.155
Store_40       3.107e+04   1.78e+04      1.750      0.080   -3741.513
```

```
6.59e+04
Store_41      -1.102e+04   6343.835     -1.737      0.082   -2.35e+04
1414.907
Store_42       1.577e+04   2.26e+04      0.697      0.486   -2.85e+04
6.01e+04
Store_43      -1.128e+05   4244.802    -26.576      0.000   -1.21e+05
-1.04e+05
Store_44      -3.554e+04   2.23e+04     -1.590      0.112   -7.93e+04
8272.512
Store_45      -5.565e+04   7919.346     -7.027      0.000   -7.12e+04
-4.01e+04
======================================================================
========
Omnibus:                     3399.229   Durbin-Watson:
2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
111741.081
Skew:                           3.007   Prob(JB):
0.00
Kurtosis:                      25.957   Cond. No.
80.5
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
"""
```

Approach: We can fix these multicollinearity with two techniques:

1.Manual Method - Variance Inflation Factor (VIF)

2.Automatic Method - Recursive Feature Elimination (RFE)

3.Feature Elmination using PCA Decomposition

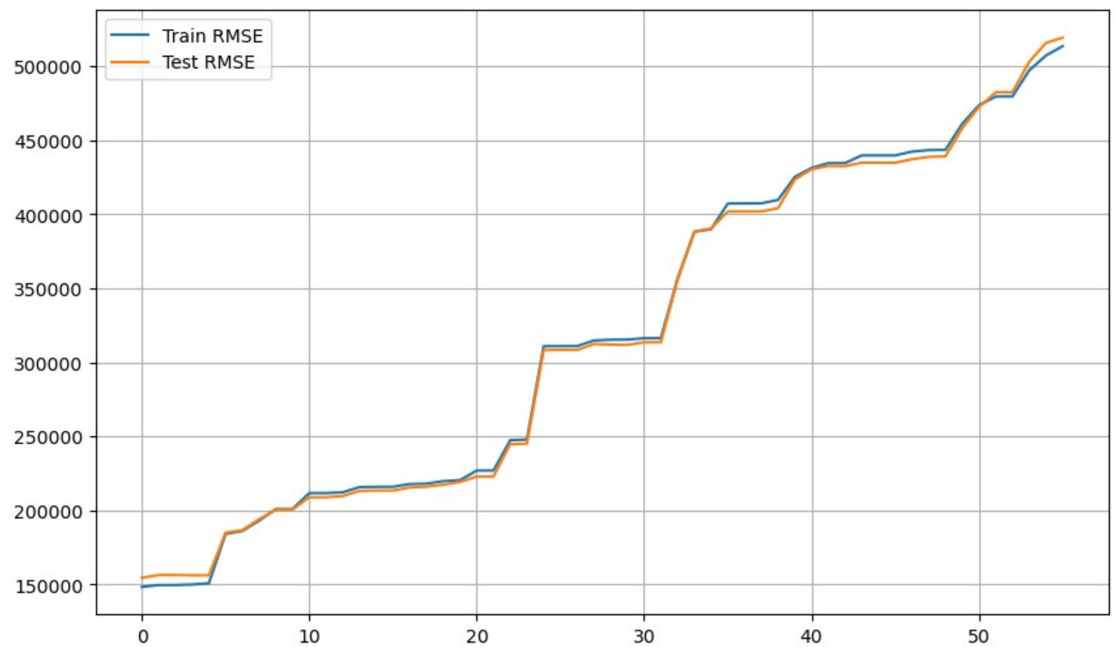
a. Manual Method - VIF

```
from sklearn.preprocessing import PolynomialFeatures
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
#Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
#Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)

DROP=[];b=[]

for i in range(len(Train_X_std.columns)):
    vif = pd.DataFrame()
    X = Train_X_std.drop(DROP,axis=1)
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in
```

```
range(X.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    vif.reset_index(drop=True, inplace=True)
    if vif.loc[0][1]>1:
        DROP.append(vif.loc[0][0])
        LR = LinearRegression()
        LR.fit(Train_X_std.drop(DROP,axis=1), Train_Y)

        pred1 = LR.predict(Train_X_std.drop(DROP,axis=1))
        pred2 = LR.predict(Test_X_std.drop(DROP,axis=1))

        Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))
        Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

        #Trd.loc[i,'ord-'+str(k)] =
round(np.sqrt(mean_squared_error(Train_Y, pred1)),2)
        #Tsd.loc[i,'ord-'+str(k)] =
round(np.sqrt(mean_squared_error(Test_Y, pred2)),2)

print('Dropped Features --> ',DROP)
#plt.plot(b)
#plt.show()
#print(API.summary())

# plt.figure(figsize=[20,4])
# plt.subplot(1,3,1)
# sns.heatmap(Trd.loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max())
# plt.title('Train RMSE')
# plt.subplot(1,3,2)
# sns.heatmap(Tsd.loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max()+10)
# plt.title('Test RMSE')
# plt.subplot(1,3,3)
# sns.heatmap((Trd+Tsd).loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max()+25)
# plt.title('Total RMSE')
# plt.show()

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.75,20.75])
plt.legend()
plt.grid()
plt.show()
```

```
Dropped Features -->  ['CPI', 'Unemployment', 'Fuel_Price',
'weekday_4', 'month_7', 'Store_7', 'Temperature', 'month_12',
'Store_43', 'year_2012', 'Store_30', 'month_2', 'month_11',
```

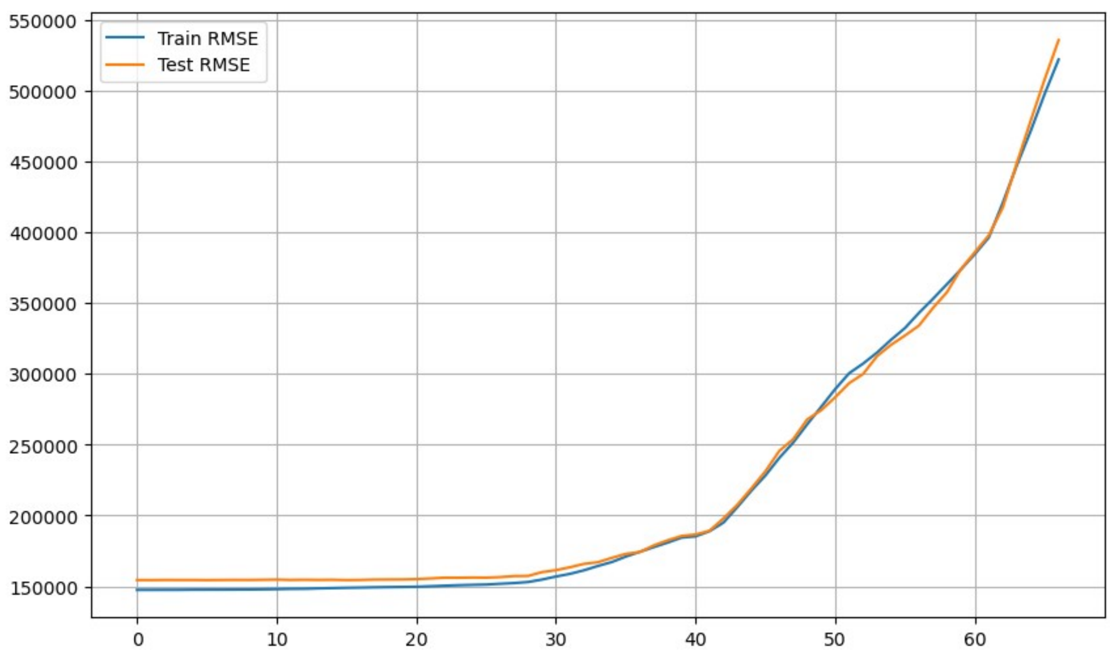
```
'Store_16', 'month_5', 'Store_25', 'Store_29', 'month_10', 'Store_17',
'Holiday_Flag', 'Store_18', 'year_2011', 'Store_19', 'month_9',
'Store_20', 'Store_8', 'Store_34', 'Store_15', 'Store_22', 'month_6',
'Store_21', 'Store_35', 'Store_14', 'Store_13', 'Store_45',
'Store_27', 'month_3', 'weekday_1', 'Store_23', 'Store_44',
'Store_42', 'Store_11', 'weekday_5', 'Store_39', 'weekday_2',
'weekday_3', 'Store_24', 'Store_41', 'Store_40', 'Store_10',
'Store_36', 'Store_9', 'month_4', 'Store_2', 'Store_3', 'Store_6']
```



b. Automatic Method - RFE

```
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)

m=df.shape[1]-2
for i in range(m):
    lm = LinearRegression()
    rfe = RFE(lm,n_features_to_select=Train_X_std.shape[1]-i)
# running RFE
    rfe = rfe.fit(Train_X_std, Train_Y)

    LR = LinearRegression()
    LR.fit(Train_X_std.loc[:,rfe.support_], Train_Y)

    pred1 = LR.predict(Train_X_std.loc[:,rfe.support_])
    pred2 = LR.predict(Test_X_std.loc[:,rfe.support_])

    Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))
    Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))
```

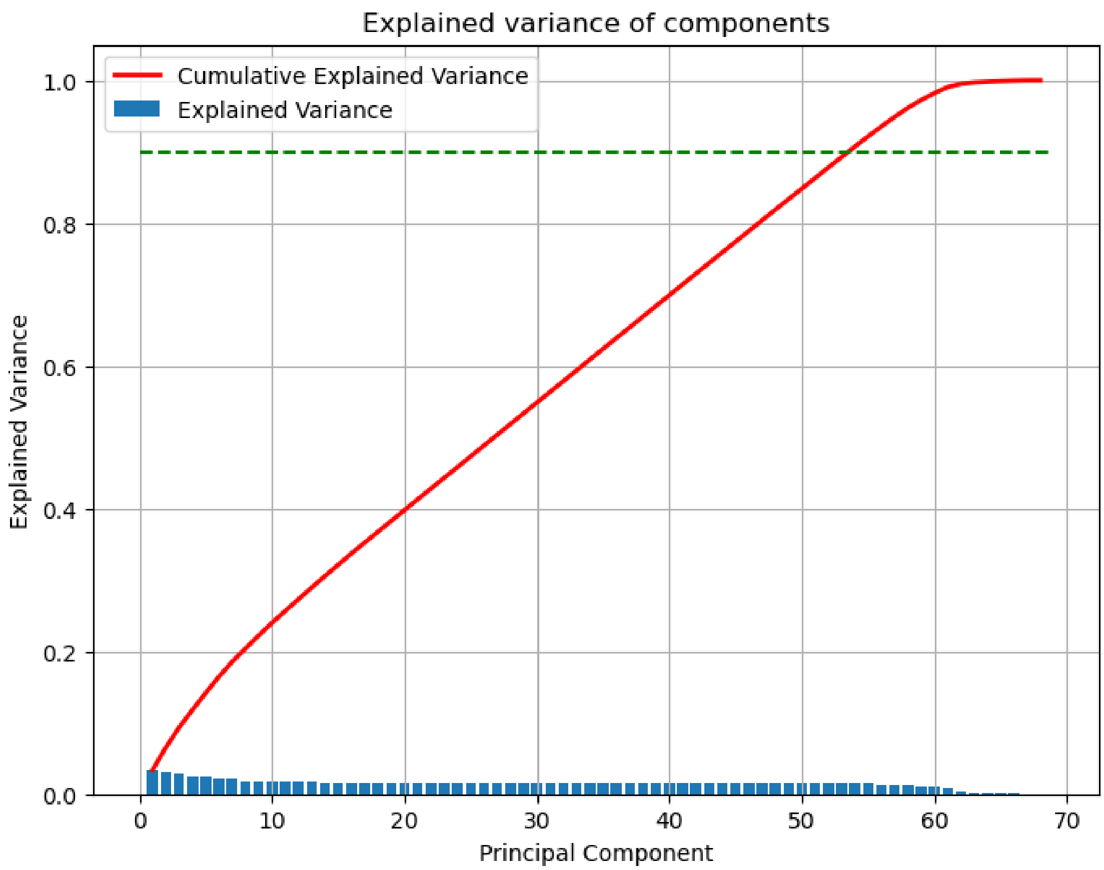
```
# plt.figure(figsize=[20,4])
# plt.subplot(1,3,1)
# sns.heatmap(Trd.loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max())
# plt.title('Train RMSE')
# plt.subplot(1,3,2)
# sns.heatmap(Tsd.loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max()+10)
# plt.title('Test RMSE')
# plt.subplot(1,3,3)
# sns.heatmap((Trd+Tsd).loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max()+25)
# plt.title('Total RMSE')
# plt.show()

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.75,20.75])
plt.legend()
plt.grid()
plt.show()
```



c. Feature Elmination using PCA Decomposition

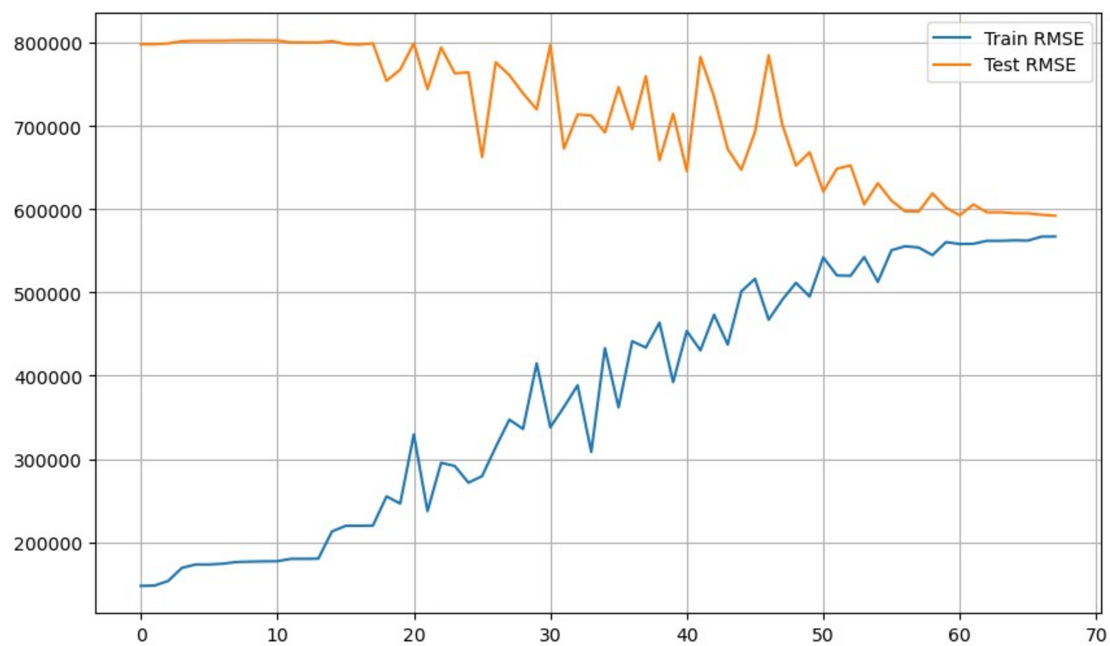
```
pca = PCA().fit(Train_X_std)

fig, ax = plt.subplots(figsize=(8,6))
x_values = range(1, pca.n_components_+1)
ax.bar(x_values, pca.explained_variance_ratio_, lw=2, label='Explained
Variance')
```

```
ax.plot(x_values, np.cumsum(pca.explained_variance_ratio_), lw=2,
label='Cumulative Explained Variance', color='red')
plt.plot([0,pca.n_components_+1],[0.9,0.9],'g--')
ax.set_title('Explained variance of components')
ax.set_xlabel('Principal Component')
ax.set_ylabel('Explained Variance')
plt.legend()
plt.grid()
plt.show()
```




```
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
m=df.shape[1]-1

for i in range(m):
    pca = PCA(n_components=Train_X_std.shape[1]-i)
    Train_X_std_pca = pca.fit_transform(Train_X_std)
    Test_X_std_pca = pca.fit_transform(Test_X_std)

    LR = LinearRegression()
    LR.fit(Train_X_std_pca, Train_Y)
```

```
    pred1 = LR.predict(Train_X_std_pca)
    pred2 = LR.predict(Test_X_std_pca)

    Trr.append(round(np.sqrt(mean_squared_error(Train_Y, pred1)),2))
    Tss.append(round(np.sqrt(mean_squared_error(Test_Y, pred2)),2))

# plt.figure(figsize=[20,4.5])
# plt.subplot(1,3,1)
# sns.heatmap(Trd.loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max())
# plt.title('Train RMSE')
# plt.subplot(1,3,2)
# sns.heatmap(Tsd.loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max()+10)
# plt.title('Test RMSE')
# plt.subplot(1,3,3)
# sns.heatmap((Trd+Tsd).loc[:6], cmap='BuGn', annot=True, vmin=0,
vmax=Trd.max().max()+25)
# plt.title('Total RMSE')
# plt.show()

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.5,20.75])
plt.legend()
plt.grid()
plt.show()
```



It can be seen that the performance of the modelsis quiet comparable unpon dropping features using VIF, RFE & PCA Techniques. Comparing the RMSE plots, the optimal values were found for dropping most features using manual RFE Technique. But let us skip these for now, as the advanced ML Algorithms take care of multicollinearity.

```
#Shortlisting the selected Features (with RFE)

lm = LinearRegression()
rfe = RFE(lm,n_features_to_select=Train_X_std.shape[1]-28)
# running RFE
rfe = rfe.fit(Train_X_std, Train_Y)

LR = LinearRegression()
LR.fit(Train_X_std.loc[:,rfe.support_], Train_Y)

#print(Train_X_std.loc[:,rfe.support_].columns)

pred1 = LR.predict(Train_X_std.loc[:,rfe.support_])
pred2 = LR.predict(Test_X_std.loc[:,rfe.support_])

print(np.sqrt(mean_squared_error(Train_Y, pred1)))
print(np.sqrt(mean_squared_error(Test_Y, pred2)))

Train_X_std = Train_X_std.loc[:,rfe.support_]
Test_X_std = Test_X_std.loc[:,rfe.support_]
```

```
152984.3455868294
157283.79051514962
```

6. Predictive Modelling

```
#Let us first define a function to evaluate our models

Model_Evaluation_Comparison_Matrix = pd.DataFrame(np.zeros([5,8]),
columns=['Train-R2','Test-R2','Train-RSS','Test-RSS',

'Train-MSE','Test-MSE','Train-RMSE','Test-RMSE'])
rc=np.random.choice(Train_X_std.loc[:,Train_X_std.nunique()>=50].colum
ns.values,2,replace=False)
def Evaluate(n, pred1,pred2):
    #Plotting predicted predicteds alongside the actual datapoints
    plt.figure(figsize=[15,6])
    for e,i in enumerate(rc):
        plt.subplot(2,3,e+1)
        plt.scatter(y=Train_Y, x=Train_X_std[i], label='Actual')
        plt.scatter(y=pred1, x=Train_X_std[i], label='Prediction')
        plt.legend()
    plt.show()

    #Evaluating the Multiple Linear Regression Model
```

```
    print('\n\n{}Training Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Training set --->',round(r2_score(Train_Y,
pred1),20))
    print('Residual Sum of Squares (RSS) on Training set  ---
>',round(np.sum(np.square(Train_Y-pred1)),20))
    print('Mean Squared Error (MSE) on Training set       ---
>',round(mean_squared_error(Train_Y, pred1),20))
    print('Root Mean Squared Error (RMSE) on Training set ---
>',round(np.sqrt(mean_squared_error(Train_Y, pred1)),20))
    print('\n{}Testing Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Testing set --->',round(r2_score(Test_Y,
pred2),20))
    print('Residual Sum of Squares (RSS) on Training set  ---
>',round(np.sum(np.square(Test_Y-pred2)),20))
    print('Mean Squared Error (MSE) on Training set       ---
>',round(mean_squared_error(Test_Y, pred2),20))
    print('Root Mean Squared Error (RMSE) on Training set ---
>',round(np.sqrt(mean_squared_error(Test_Y, pred2)),20))
    print('\n{}Residual Plots{}'.format('-'*20, '-'*20))

    Model_Evaluation_Comparison_Matrix.loc[n,'Train-R2']  =
round(r2_score(Train_Y, pred1),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-R2']   =
round(r2_score(Test_Y, pred2),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RSS'] =
round(np.sum(np.square(Train_Y-pred1)),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RSS']  =
round(np.sum(np.square(Test_Y-pred2)),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-MSE'] =
round(mean_squared_error(Train_Y, pred1),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-MSE']  =
round(mean_squared_error(Test_Y, pred2),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RMSE']=
round(np.sqrt(mean_squared_error(Train_Y, pred1)),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RMSE'] =
round(np.sqrt(mean_squared_error(Test_Y, pred2)),20)

    # Plotting y_test and y_pred to understand the spread.
    plt.figure(figsize=[15,4])

    plt.subplot(1,2,1)
    sns.distplot((Train_Y - pred1))
    plt.title('Error Terms')
    plt.xlabel('Errors')

    plt.subplot(1,2,2)
    plt.scatter(Train_Y,pred1)
    plt.plot([Train_Y.min(),Train_Y.max()],
[Train_Y.min(),Train_Y.max()], 'r--')
    plt.title('Test vs Prediction')
```

```
plt.xlabel('y_test')
plt.ylabel('y_pred')
plt.show()
```

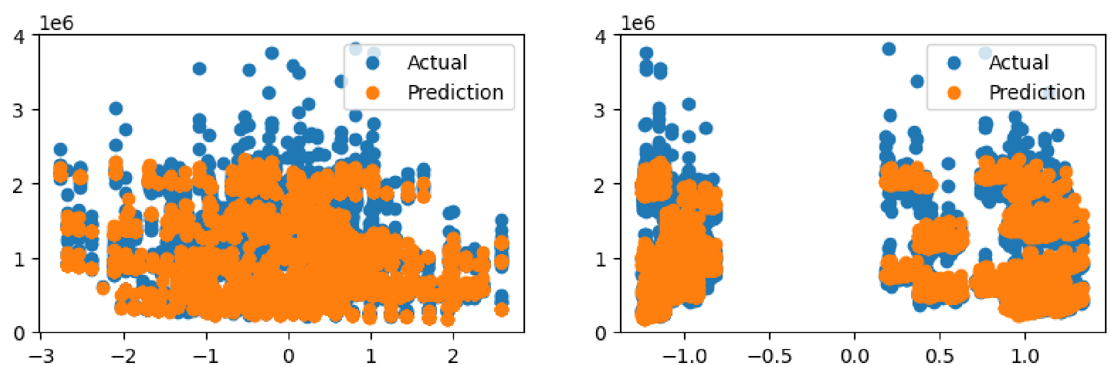
a. Multiple Linear Regression(MLR)

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ \dots + \beta_i X_i$$

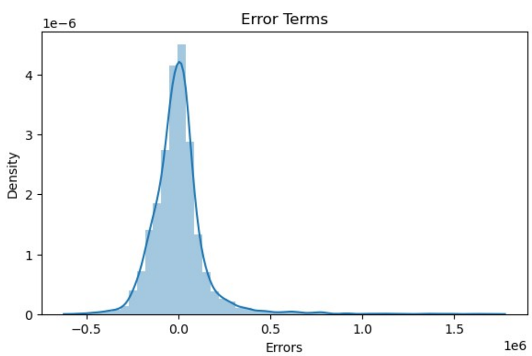
Y : Dependent variable
β_0 : Intercept
β_i : Slope for X_i
X = Independent variable

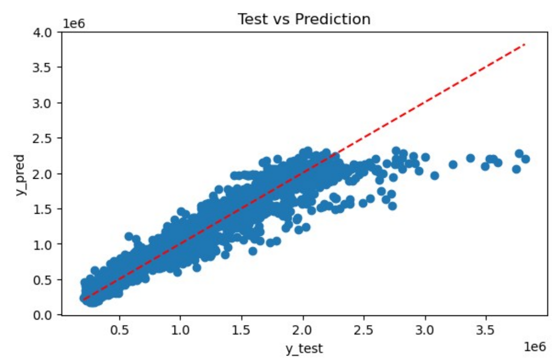
```
#Linear Regression

MLR = LinearRegression().fit(Train_X_std,Train_Y)
pred1 = MLR.predict(Train_X_std)
pred2 = MLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Multiple Linear Regression Model \
033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
#print('The Coeffecient of the Regresion Model was found to be
',MLR.coef_)
print('The Intercept of the Regresion Model was found to be
',MLR.intercept_)

Evaluate(0, pred1, pred2)
```

```
<<<----------------------------------- Evaluating Multiple Linear
Regression Model ----------------------------------->>>

The Intercept of the Regresion Model was found to be
1047603.298112138
```



```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.9276826744775732
Residual Sum of Squares (RSS) on Training set  ---> 111450847994430.22
Mean Squared Error (MSE) on Training set       ---> 23404209994.630455
Root Mean Squared Error (RMSE) on Training set ---> 152984.3455868294

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.927676279121959
Residual Sum of Squares (RSS) on Training set  ---> 29463185193746.844
Mean Squared Error (MSE) on Training set       ---> 24738190758.81347
Root Mean Squared Error (RMSE) on Training set ---> 157283.79051514962

--------------------Residual Plots--------------------
```





b. Ridge Regression Model

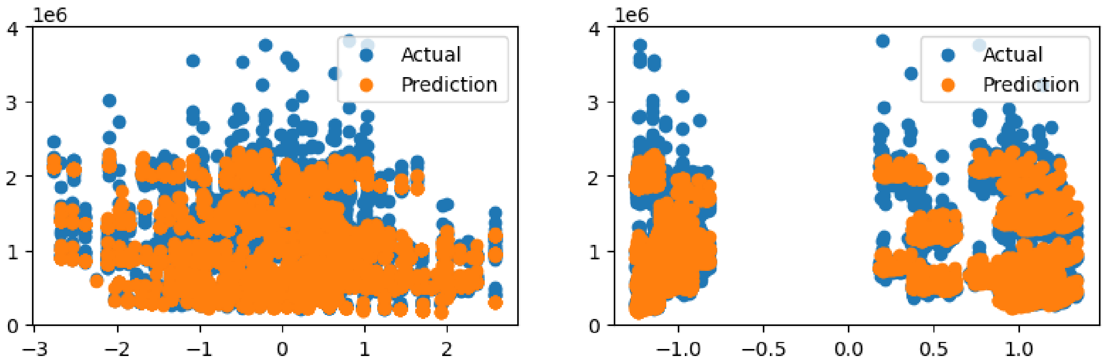
Ridge Formula: Sum of Error + Sum of the squares of coefficients

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda\sum\beta^2$$

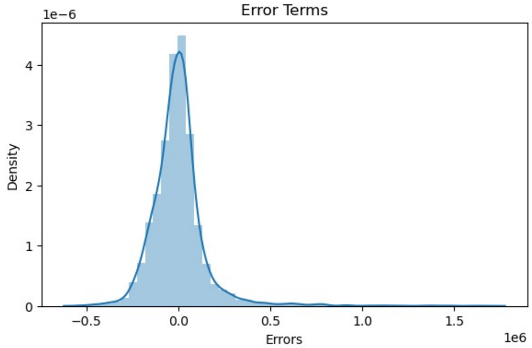
```
#Creating a Ridge Regression model

RLR = Ridge().fit(Train_X_std,Train_Y)
pred1 = RLR.predict(Train_X_std)
pred2 = RLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Ridge Regression Model \033[0m{}{}\
```

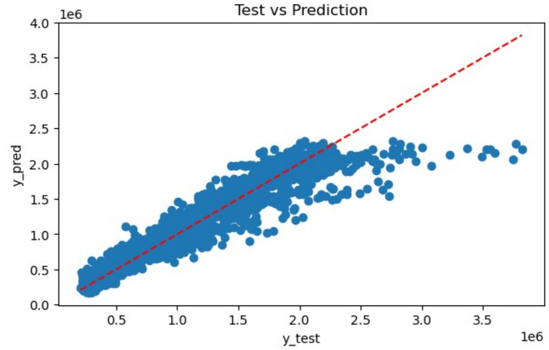
```
n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
#print('The Coeffecient of the Regresion Model was found to be
',MLR.coef_)
print('The Intercept of the Regresion Model was found to be
',MLR.intercept_)

Evaluate(1, pred1, pred2)
```

```
<<<----------------------------------- Evaluating Ridge Regression
Model ----------------------------------->>>

The Intercept of the Regresion Model was found to be
1047603.298112138
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.9276821973327433
Residual Sum of Squares (RSS) on Training set  ---> 111451583339598.69
Mean Squared Error (MSE) on Training set       ---> 23404364414.027443
Root Mean Squared Error (RMSE) on Training set ---> 152984.8502761873

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.927696636618113
Residual Sum of Squares (RSS) on Training set  ---> 29454891971661.723
Mean Squared Error (MSE) on Training set       ---> 24731227516.088768
Root Mean Squared Error (RMSE) on Training set ---> 157261.65303750552

--------------------Residual Plots--------------------
```



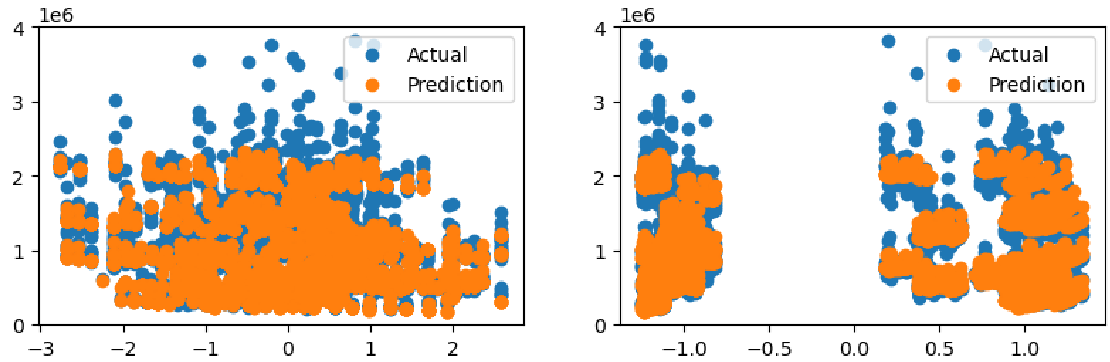


c. Lasso Regression Model

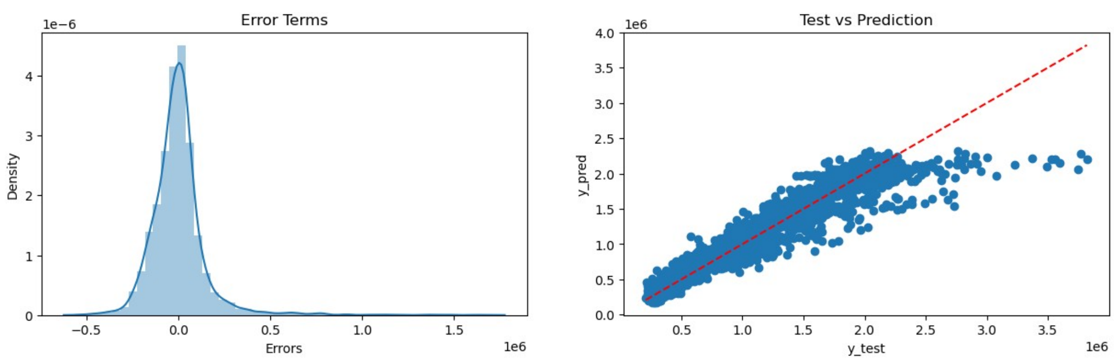
Lasso = Sum of Error + Sum of the absolute value of coefficients

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda \sum |\beta|$$

```
#Creating a Ridge Regression model

LLR = Lasso().fit(Train_X_std,Train_Y)
pred1 = LLR.predict(Train_X_std)
pred2 = LLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Lasso Regression Model \033[0m{}{}\
n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
#print('The Coeffecient of the Regresion Model was found to be
',MLR.coef_)
print('The Intercept of the Regresion Model was found to be
',MLR.intercept_)

Evaluate(2, pred1, pred2)

<<<----------------------------------- Evaluating Lasso Regression
Model ----------------------------------->>>

The Intercept of the Regresion Model was found to be
1047603.298112138
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.9276826740433101
Residual Sum of Squares (RSS) on Training set  ---> 111450848663688.89
Mean Squared Error (MSE) on Training set       ---> 23404210135.171963
Root Mean Squared Error (RMSE) on Training set ---> 152984.3460461624

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.9276767498337136
Residual Sum of Squares (RSS) on Training set  ---> 29462993435532.15
Mean Squared Error (MSE) on Training set       ---> 24738029752.75579
Root Mean Squared Error (RMSE) on Training set ---> 157283.27868135186

--------------------Residual Plots--------------------
```



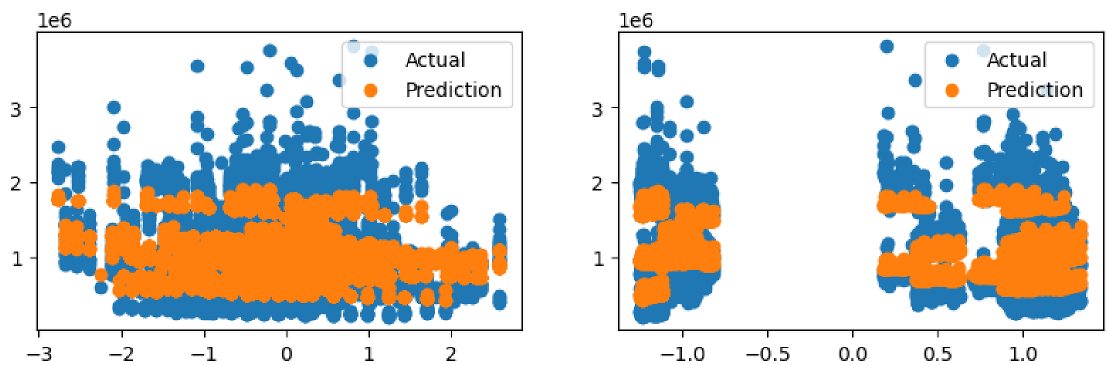
d. Elastic-Net Regression

Elastic Net Formula: Ridge + Lasso

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda\sum\beta^2 + \lambda\sum|\beta|$$

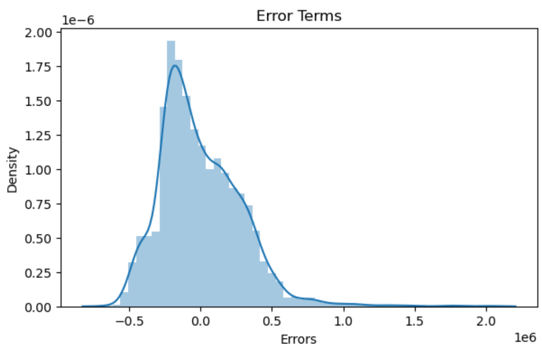
```
#Creating a ElasticNet Regression model

ENR = ElasticNet().fit(Train_X_std,Train_Y)
pred1 = ENR.predict(Train_X_std)
pred2 = ENR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Elastic-Net Regression Model \033[0m{}
{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
#print('The Coeffecient of the Regresion Model was found to be
',MLR.coef_)
print('The Intercept of the Regresion Model was found to be
',MLR.intercept_)

Evaluate(3, pred1, pred2)
```

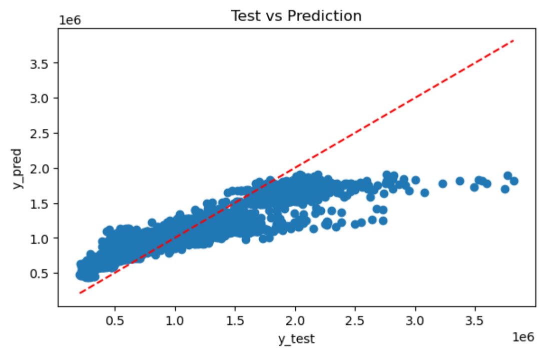
```
<<<----------------------------------- Evaluating Elastic-Net
Regression Model ----------------------------------->>>

The Intercept of the Regresion Model was found to be
1047603.298112138
```




```
--------------------Training Set Metrics--------------------
```

```
R2-Score on Training set ---> 0.7477826893125256
Residual Sum of Squares (RSS) on Training set  ---> 388701226876488.9
Mean Squared Error (MSE) on Training set       ---> 81625625131.56003
Root Mean Squared Error (RMSE) on Training set ---> 285701.98657265236

--------------------Testing Set Metrics-------------------

R2-Score on Testing set ---> 0.7599512663907991
Residual Sum of Squares (RSS) on Training set  ---> 97790879783118.03
Mean Squared Error (MSE) on Training set       ---> 82108211404.80104
Root Mean Squared Error (RMSE) on Training set ---> 286545.3042797963

--------------------Residual Plots--------------------
```





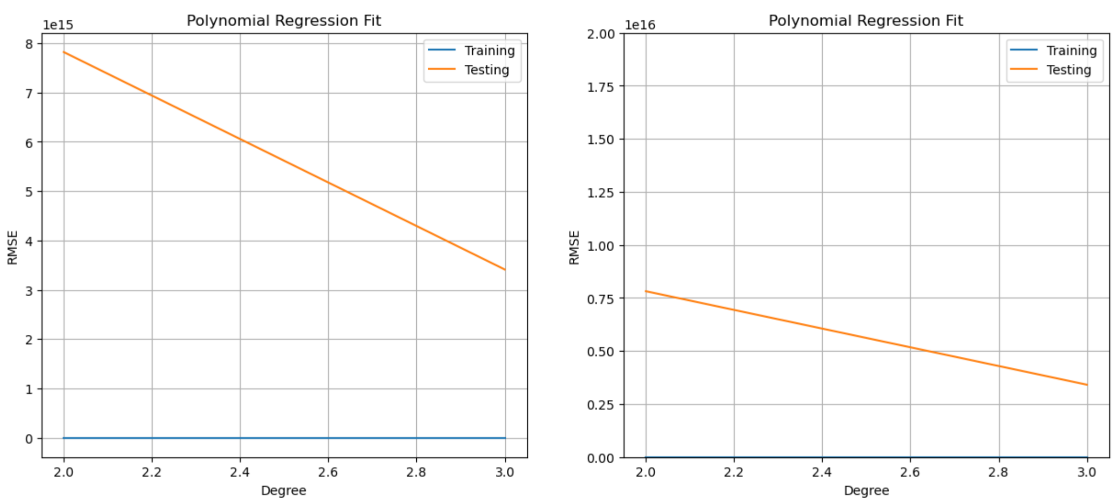
e. Polynomial Regression Model

Polynomial Regression : Order-n

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

```
#Checking polynomial regression performance on various degrees

Trr=[]; Tss=[]
n_degree=4

for i in range(2,n_degree):
    #print(f'{i} Degree')
    poly_reg = PolynomialFeatures(degree=i)
    X_poly = poly_reg.fit_transform(Train_X_std)
    X_poly1 = poly_reg.fit_transform(Test_X_std)
    LR = LinearRegression()
    LR.fit(X_poly, Train_Y)
    pred1 = LR.predict(X_poly)
```

```
    Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))

    pred2 = LR.predict(X_poly1)
    Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

plt.figure(figsize=[15,6])
plt.subplot(1,2,1)
plt.plot(range(2,n_degree),Trr, label='Training')
plt.plot(range(2,n_degree),Tss, label='Testing')
#plt.plot([1,4],[1,4],'b--')
plt.title('Polynomial Regression Fit')
#plt.ylim([0,5])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
#plt.xticks()

plt.subplot(1,2,2)
plt.plot(range(2,n_degree),Trr, label='Training')
plt.plot(range(2,n_degree),Tss, label='Testing')
plt.title('Polynomial Regression Fit')
plt.ylim([0,2e16])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
#plt.xticks()
plt.show()
```

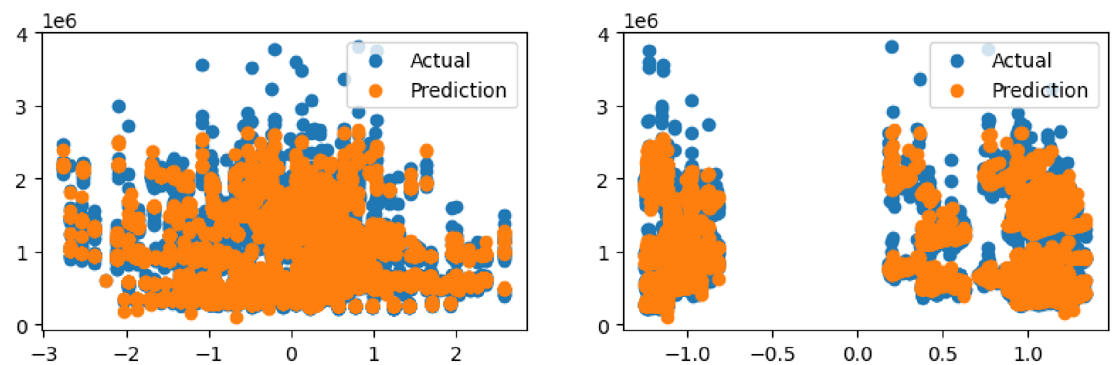


choosing 2nd order polynomial regression to get the optimal training & testing scores

```
#Using the 2nd Order Polynomial Regression model (degree=2)
poly_reg = PolynomialFeatures(degree=2)
X_poly = poly_reg.fit_transform(Train_X_std)
X_poly1 = poly_reg.fit_transform(Test_X_std)
```

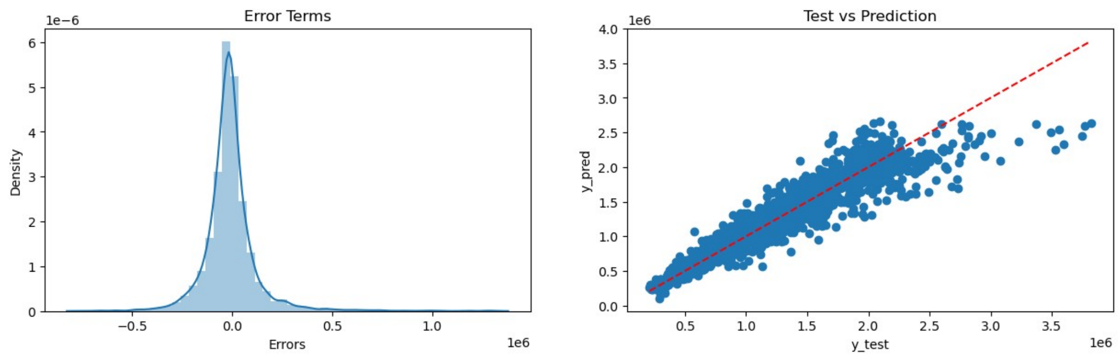
```
PR = LinearRegression()
PR.fit(X_poly, Train_Y)

pred1 = PR.predict(X_poly)
pred2 = PR.predict(X_poly1)

print('{}{}\033[1m Evaluating Polynomial Regression Model \033[0m{}{}\
n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be
',MLR.coef_)
print('The Intercept of the Regresion Model was found to be
',MLR.intercept_)

Evaluate(4, pred1, pred2)
```

```
<<<----------------------------------- Evaluating Polynomial
Regression Model ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be
[ 382416.72707697  -87749.08819447  -33466.21811552  -70608.43596712
   23371.88280837   44272.12738693   67885.30758384 -171730.97447116
  194110.33061399 -182942.03065485  -95260.47050719 -110787.17299967
 -154257.07523525  200572.26744363  -27802.83358985   36027.35653929
  202356.39957846  137960.5727378  -126652.1156391    29819.45548371
   75864.89398192  119328.98106792  104372.22301835 -108321.57336704
   54083.53671998   65645.06638736  110781.93941655 -114111.68634244
   52174.80531825  169276.22877422   47325.81863416 -164944.63849199
  -45218.40228284   78308.2717848    42369.70043942 -163871.15661923
 -143446.53544316  -94113.42272716  -57253.08807135  -49965.33157002]
The Intercept of the Regresion Model was found to be
1047603.298112138
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.942947379382929
Residual Sum of Squares (RSS) on Training set  ---> 87925858736371.45
```

```
Mean Squared Error (MSE) on Training set       ---> 18464061053.416935
Root Mean Squared Error (RMSE) on Training set ---> 135882.526667033

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> -2.499590740902291e+19
Residual Sum of Squares (RSS) on Training set  --->
1.0182814713303774e+34
Mean Squared Error (MSE) on Training set       --->
8.549802446098889e+30
Root Mean Squared Error (RMSE) on Training set ---> 2924004522243234.5

--------------------Residual Plots--------------------
```



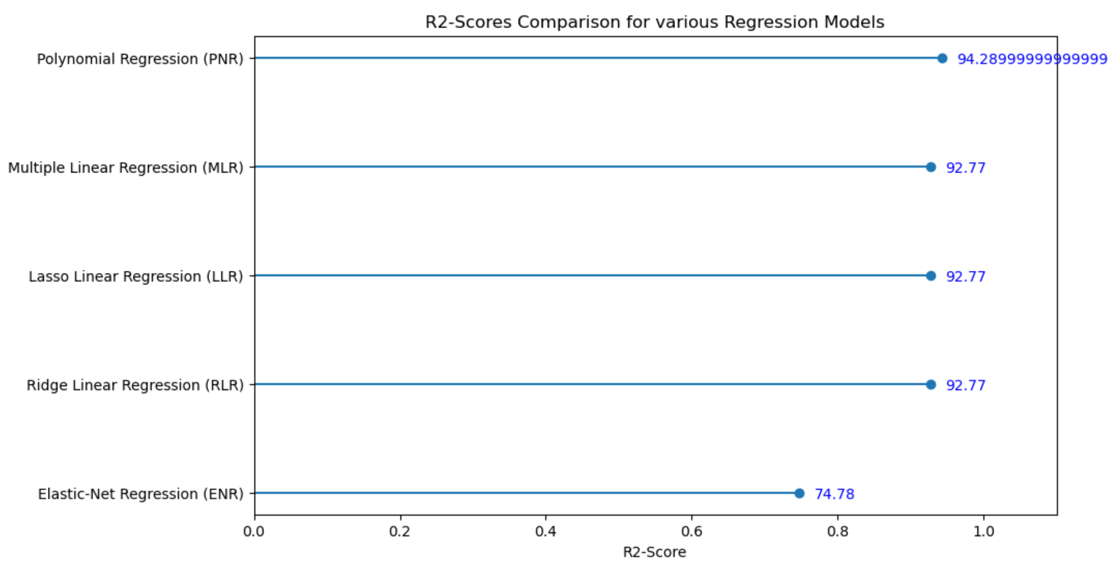
f. Comparing the Evaluation Metics of the Models

```
# Regression Models Results Evaluation

EMC = Model_Evaluation_Comparison_Matrix.copy()
EMC.index = ['Multiple Linear Regression (MLR)','Ridge Linear
Regression (RLR)','Lasso Linear Regression (LLR)','Elastic-Net
Regression (ENR)','Polynomial Regression (PNR)']
EMC
```

```
                                  Train-R2       Test-R2     Train-RSS
\
Multiple Linear Regression (MLR)  0.927683  9.276763e-01  1.114508e+14

Ridge Linear Regression (RLR)     0.927682  9.276966e-01  1.114516e+14

Lasso Linear Regression (LLR)     0.927683  9.276767e-01  1.114508e+14

Elastic-Net Regression (ENR)      0.747783  7.599513e-01  3.887012e+14

Polynomial Regression (PNR)       0.942947 -2.499591e+19  8.792586e+13


                                      Test-RSS     Train-MSE
Test-MSE  \
```

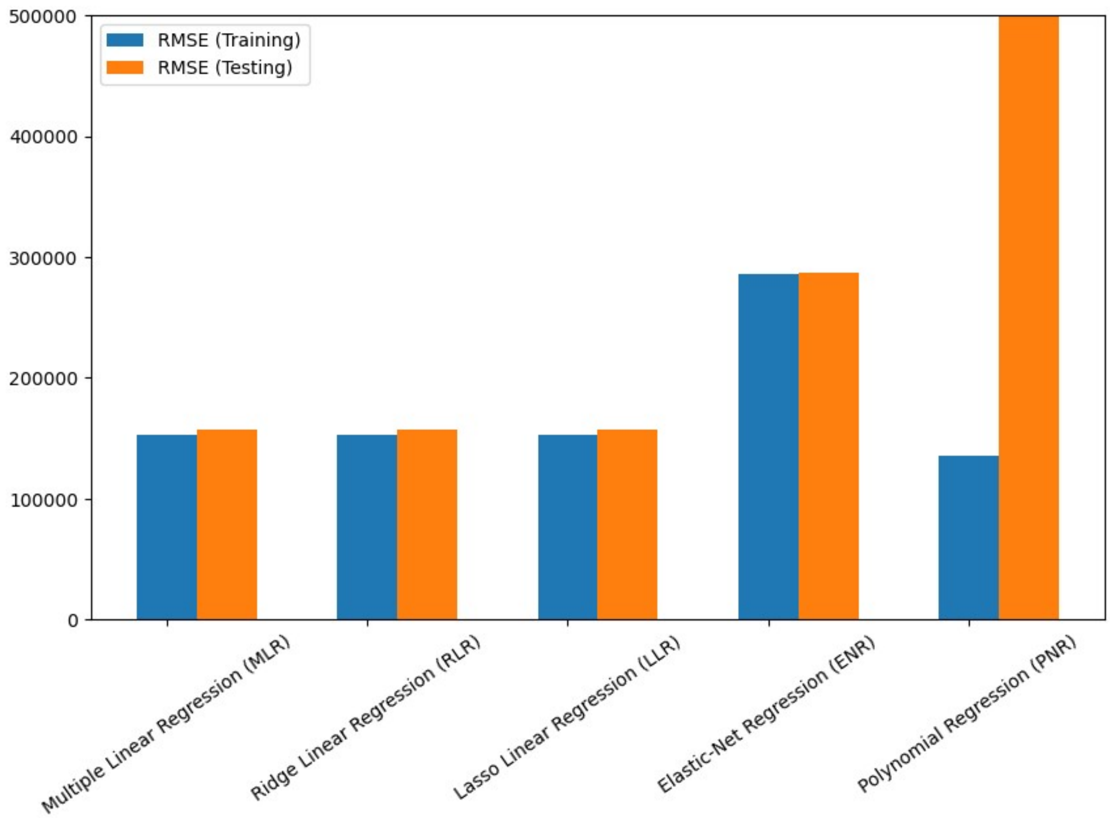
```
Multiple Linear Regression (MLR)  2.946319e+13  2.340421e+10
2.473819e+10
Ridge Linear Regression (RLR)     2.945489e+13  2.340436e+10
2.473123e+10
Lasso Linear Regression (LLR)     2.946299e+13  2.340421e+10
2.473803e+10
Elastic-Net Regression (ENR)      9.779088e+13  8.162563e+10
8.210821e+10
Polynomial Regression (PNR)       1.018281e+34  1.846406e+10
8.549802e+30

                                     Train-RMSE     Test-RMSE
Multiple Linear Regression (MLR)  152984.345587  1.572838e+05
Ridge Linear Regression (RLR)     152984.850276  1.572617e+05
Lasso Linear Regression (LLR)     152984.346046  1.572833e+05
Elastic-Net Regression (ENR)      285701.986573  2.865453e+05
Polynomial Regression (PNR)       135882.526667  2.924005e+15
```

```
# R2-Scores Comparison for different Regression Models

R2 = round(EMC['Train-R2'].sort_values(ascending=True),4)
plt.hlines(y=R2.index, xmin=0, xmax=R2.values)
plt.plot(R2.values, R2.index,'o')
plt.title('R2-Scores Comparison for various Regression Models')
plt.xlabel('R2-Score')
#plt.ylabel('Regression Models')
for i, v in enumerate(R2):
    plt.text(v+0.02, i-0.05, str(v*100), color='blue')
plt.xlim([0,1.1])
plt.show()
```



it is clear that the polynomial regresion models have the highest explainability power to understand the dataset.

```
# Root Mean SquaredError Comparison for different Regression Models

cc = Model_Evaluation_Comparison_Matrix.columns.values
s=5

plt.bar(np.arange(5),
Model_Evaluation_Comparison_Matrix[cc[6]].values, width=0.3,
label='RMSE (Training)')
plt.bar(np.arange(5)+0.3,
Model_Evaluation_Comparison_Matrix[cc[7]].values, width=0.3,
label='RMSE (Testing)')
plt.xticks(np.arange(5),EMC.index, rotation =35)
plt.legend()
plt.ylim([0,500000])
plt.show()
```



Lesser the RMSE, better the model! Also, provided the model should have close proximity with the training & testing scores. For this problem, it is can be said that polynomial regressions clearly overfitting the current problem. Here simple Multiple Linear Regression Model gave the best results.

7. Project Outcomes & Conclusions